**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Верификация распределенных алгоритмов»**

Тема: Разработка контроллера светофоров и его верификация

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Колованов Р.А. |
| Преподаватель |  | Шошмина И.В. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Колованов Р.А. | | |
| Группа 9303 | | |
| Тема работы: Разработка контроллера светофоров и его верификация | | |
| Исходные данные:   * Выдается несколько пересечений на сложном перекрестке; * Необходимо составить модель контроллера светофора, управляющего движением автомобилей по перекрестку:   + Машины на каждом направлении движутся независимо;   + Появление машины в каждом направлении регистрируется своим независимым датчиком движения;   + Контроллер светофора в каждом направлении работает по алгоритму, данному в методичке для другого перекрестка; * Верифицировать ее относительно заданных требований. | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Ход работа», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: 11.04.2024 | | |
| Дата сдачи реферата: 06.06.2024 | | |
| Дата защиты реферата: 06.06.2024 | | |
| Студент |  | Колованов Р.А. |
| Преподаватель |  | Шошмина И.В. |

**Аннотация**

В данной курсовой работе рассматривается задача разработки контроллера светофора для управления движением автомобилей на сложном перекрестке. Была разработана модель контроллера светофоров на языке Promela. Разработанная модель была верифицирована по трем свойствам: безопасность, живость и справедливость.

**Summary**

In this course work, the task of developing a traffic light controller for controlling the movement of cars at a complex intersection is considered. A traffic lights controller model has been developed in the Promela language. The developed model was verified according to three properties: safety, liveliness and fairness.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Разработка модели | 6 |
| 1.1. | Задание | 6 |
| 1.2. | Описание состояний модели | 7 |
| 1.3. | Описание процессов модели | 7 |
| 2. | Верификация модели | 9 |
| 2.1. | Верификация свойства безопасности | 9 |
| 2.2. | Верификация свойства живости | 9 |
| 2.3. | Верификация свойства справедливости | 10 |
|  | Заключение | 11 |
|  | Список использованных источников | 12 |
|  | Приложение А. Модель на языке Promela | 13 |

**введение**

Проверка корректности работы распределенных алгоритмов зачастую является крайне трудоемкой задачей, в которой требуется рассмотреть все возможные состояния системы. Для решения данной задачи были разработаны специальные инструменты – верификаторы, которые на основании разработанной модели системы проверяют выполнимость заданных требований. В данной работе рассматривается язык Promela для моделирования распределенных систем и верификатор Spin.

Целью работы является разработка модели контроллера светофоров на перекрестке, которая будет обрабатывать недетерминированный поток машин в соответствии с заданными свойствами. Для разработки и верификации модели распределенного алгоритма контроллера светофоров необходимо обойтись без предположений об очередности действий и дать возможность контроллерам движения и контроллерам светофоров действовать независимо друг от друга в борьбе за общие ресурсы. Также необходимо учесть, что перекресток могут пересекать несколько потоков машин одновременно, если направления движения не пересекаются.

**1. РАЗРАБОТКА МОДЕЛИ**

**1.1. Задание**

Вариант 5. Пересечения SD/WN, SD/DN, WN/DE, NS/WE (24, 34, 43, 5).

Суммарно на сложном перекрестке имеется 6 направлений движения (SD, WN, DN, DE, NS, WE), которые образуют 10 пересечений друг с другом.

Схема сложного перекрестка с направлениями движения и точками пересечения представлена на рис. 1.

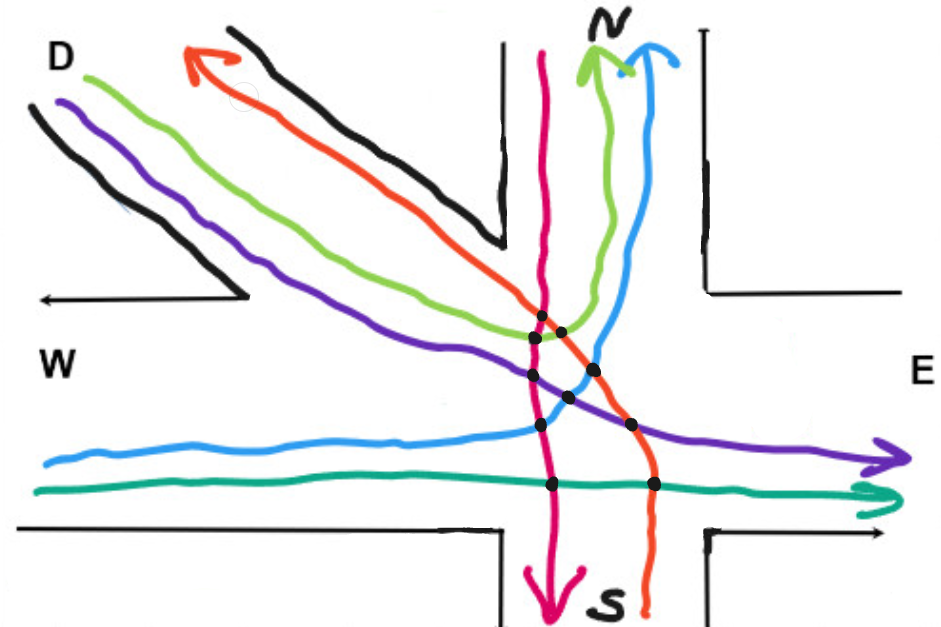


Рисунок 1 – Схема сложного перекрестка с направлениями движения и точками пересечения

Направление SD пересекается с направлениями WE, DE, WN, DN, NS; направление WN – с NS, DE, SD; направление DN – с NS, SD; направление DE – с NS, WN, SD; направление NS – с SD, DN, DE, WN, WE; направление WE – с NS, SD.

**1.2. Описание состояний модели**

Для разработки модели необходимо выделить переменные, которые будут формировать состояния модели, то есть определять текущую ситуацию на перекрестке. Состояние каждого из шести направлений сложного перекрестка было представлено в виде двух переменных:

* Переменная «XX\_SENSE», где XX – направление. Является логической переменной и хранит информацию о наличии потока машин в соответствующем направлении;
* Переменная «XX\_LIGHT», где XX – направление. Является целочисленной переменной и хранит информацию о текущем цвете света светофора. Символьная константа RED означает, что горит красный свет, символьная константа GREEN – что зеленый свет.

Для синхронизации процессов были введены логические переменные «XX\_LOCK», где XX – направление. Если «XX\_LOCK» равен true, то процесс контроллера направления XX захватил ресурс и может пускать машины по своему направлению через перекресток.

**1.3. Описание процессов модели**

Для модели были разработаны шесть процессов ControllerXX, которые представляют собой независимые контроллеры светофора на соответствующих направлениях перекрестка (XX – направление), и процесс EnvironmentController, который представляет собой внешнюю среду. В завершение был написан главный процесс init, осуществляющий запуск процессов контроллеров и процесса внешней среды атомарно.

Процесс EnvironmentController представляет собой бесконечный цикл, который отвечает за работу датчиков наличия машины на всех направлениях (XX\_SENSE). Если на направлении нет машин и включен красный свет светофора, то на направлении может появиться поток машин. Если же на направлении есть поток машин и горит зеленый свет светофора (т.е. машины в данный момент движутся через перекресток), то на направлении поток машин может закончиться.

Все процессы контроллеров светофоров ControllerXX схожи по реализации. В первом условии они проверяют, есть ли запрос на проезд в их направлении и не горит ли уже зеленый свет светофора. Если это так, то атомарно делается проверка на отсутствие блокировок на конфликтующих направлениях, после чего устанавливается блокировка и включает зеленый свет. Считается, что в этот момент машины начинают проезжать в направлении. Вторым условием является наличие включенного зеленого света светофора и отсутствие запроса на проезд. Если оно выполнено, сначала выключается свет, а затем снимается блокировка. Порядок важен для безопасности.

Полный код модели на Promela представлен в Приложении А.

**2. ВЕРИФИКАЦИЯ МОДЕЛИ**

Для верификации модели было проверено выполнение свойств безопасности, живости и справедливости для каждого направления.

**2.1. Верификация свойства безопасности**

Свойство безопасности означает, что никогда не будет разрешен проезд в пересекающихся направлениях.

Свойство безопасности для направлений были представлены следующими формулами:

safetySD ::= G !((SD\_LIGHT == GREEN) && (WE\_LIGHT == GREEN || DE\_LIGHT == GREEN || WN\_LIGHT == GREEN || DN\_LIGHT == GREEN || NS\_LIGHT == GREEN))

safetyWN ::= G !((WN\_LIGHT == GREEN) && (DE\_LIGHT == GREEN || SD\_LIGHT == GREEN || NS\_LIGHT == GREEN))

safetyDN ::= G !((DN\_LIGHT == GREEN) && (SD\_LIGHT == GREEN || NS\_LIGHT == GREEN))

safetyDE ::= G !((DE\_LIGHT == GREEN) && (SD\_LIGHT == GREEN || WN\_LIGHT == GREEN || NS\_LIGHT == GREEN))

safetyNS ::= G !((NS\_LIGHT == GREEN) && (WE\_LIGHT == GREEN || DE\_LIGHT == GREEN || WN\_LIGHT == GREEN || DN\_LIGHT == GREEN || SD\_LIGHT == GREEN))

safetyWE ::= G !((WE\_LIGHT == GREEN) && (SD\_LIGHT == GREEN || NS\_LIGHT == GREEN))

Результаты верификации подтвердили выполнение свойства безопасности для каждого направления.

**2.2. Верификация свойства живости**

Свойство живости означает, что при появлении машины ей всегда предоставится возможность проезда в нужном направлении (возможно, не сразу).

Свойство живости для направлений были представлены следующими формулами:

livenessSD ::= G((SD\_SENSE && SD\_LIGHT == RED) -> F(SD\_LIGHT == GREEN))

livenessWN ::= G ((WN\_SENSE && WN\_LIGHT == RED) -> F(WN\_LIGHT == GREEN))

livenessDN ::= G ((DN\_SENSE && DN\_LIGHT == RED) -> F(DN\_LIGHT == GREEN))

livenessDE ::= G ((DE\_SENSE && DE\_LIGHT == RED) -> F(DE\_LIGHT == GREEN))

livenessNS ::= G ((NS\_SENSE && NS\_LIGHT == RED) -> F(NS\_LIGHT == GREEN))

livenessWE ::= G ((WE\_SENSE && WE\_LIGHT == RED) -> F(WE\_LIGHT == GREEN))

Результаты верификации не подтвердили выполнение свойства живости для какого-либо направления. Были найдены контрпримеры. Это можно объяснить наличием голодания.

**2.3. Верификация свойства справедливости**

Свойство справедливости означает, что на каждом направлении не движется непрерывный поток машин.

Свойство справедливости для направлений были представлены следующими формулами:

fairnessSD ::= GF !(SD\_LIGHT == GREEN && SD\_SENSE)

fairnessWN ::= GF !(WN\_LIGHT == GREEN && WN\_SENSE)

fairnessDN ::= GF !(DN\_LIGHT == GREEN && DN\_SENSE)

fairnessDE ::= GF !(DE\_LIGHT == GREEN && DE\_SENSE)

fairnessNS ::= GF !(NS\_LIGHT == GREEN && NS\_SENSE)

fairnessWE ::= GF !(WE\_LIGHT == GREEN && WE\_SENSE)

Результаты верификации не подтвердили выполнение свойства справедливости для некоторых направлений. Были найдены контрпримеры. Это можно объяснить наличием голодания.

**заключение**

В результате выполнения работы была разработана модель контроллера светофоров на сложном перекрестке. Взятый за основу алгоритм из методических указаний для простого перекрестка был доработан. В модели использована общая память процессов для хранения состояний контроллеров, продуманы ограничения системы для соблюдения свойства безопасности. Выполнена верификация модели по критериям безопасности, живности и справедливости, выраженным в LTL-формулах.

**список использованных источников**

1. Карпов Ю. Г., Шошмина И. В. Верификация распределенных систем: учебное пособие для студентов высших учебных заведений, обучающихся по направлению подготовки магистров в образовательной области «Информатика и вычислительная техника». – 2011.

2. SPIN Verifier’s Roadmap: Spin // spinroot.com. URL: https://spinroot.com/spin/Man/4\_SpinVerification.html (дата обращения: 04.06.2024).

3. Concise Promela Reference // spinroot.com. URL: http://spinroot.com/spin/Man/Quick.html (дата обращения: 04.06.2024).

4. Basic Spin Manual // spinroot.com. URL: https://spinroot.com/spin/Man/Manual.html (дата обращения: 04.06.2024).

**приложение А**

**модель на языке promela**

// LTL

ltl safetySD { [] !((SD\_LIGHT == GREEN) && (WE\_LIGHT == GREEN || DE\_LIGHT == GREEN || WN\_LIGHT == GREEN || DN\_LIGHT == GREEN || NS\_LIGHT == GREEN)) };

ltl safetyWN { [] !((WN\_LIGHT == GREEN) && (DE\_LIGHT == GREEN || SD\_LIGHT == GREEN || NS\_LIGHT == GREEN)) };

ltl safetyDN { [] !((DN\_LIGHT == GREEN) && (SD\_LIGHT == GREEN || NS\_LIGHT == GREEN)) };

ltl safetyDE { [] !((DE\_LIGHT == GREEN) && (SD\_LIGHT == GREEN || WN\_LIGHT == GREEN || NS\_LIGHT == GREEN)) };

ltl safetyNS { [] !((NS\_LIGHT == GREEN) && (WE\_LIGHT == GREEN || DE\_LIGHT == GREEN || WN\_LIGHT == GREEN || DN\_LIGHT == GREEN || SD\_LIGHT == GREEN)) };

ltl safetyWE { [] !((WE\_LIGHT == GREEN) && (SD\_LIGHT == GREEN || NS\_LIGHT == GREEN)) };

ltl livenessSD { [] ((SD\_SENSE && SD\_LIGHT == RED) -> <> (SD\_LIGHT == GREEN)) };

ltl livenessWN { [] ((WN\_SENSE && WN\_LIGHT == RED) -> <> (WN\_LIGHT == GREEN)) };

ltl livenessDN { [] ((DN\_SENSE && DN\_LIGHT == RED) -> <> (DN\_LIGHT == GREEN)) };

ltl livenessDE { [] ((DE\_SENSE && DE\_LIGHT == RED) -> <> (DE\_LIGHT == GREEN)) };

ltl livenessNS { [] ((NS\_SENSE && NS\_LIGHT == RED) -> <> (NS\_LIGHT == GREEN)) };

ltl livenessWE { [] ((WE\_SENSE && WE\_LIGHT == RED) -> <> (WE\_LIGHT == GREEN)) };

ltl fairnessSD { [] <> !(SD\_LIGHT == GREEN && SD\_SENSE) };

ltl fairnessWN { [] <> !(WN\_LIGHT == GREEN && WN\_SENSE) };

ltl fairnessDN { [] <> !(DN\_LIGHT == GREEN && DN\_SENSE) };

ltl fairnessDE { [] <> !(DE\_LIGHT == GREEN && DE\_SENSE) };

ltl fairnessNS { [] <> !(NS\_LIGHT == GREEN && NS\_SENSE) };

ltl fairnessWE { [] <> !(WE\_LIGHT == GREEN && WE\_SENSE) };

// Traffic lights for direaction: SD, WN, DN, DE, NS, WE

mtype:light = {RED, GREEN};

mtype:light SD\_LIGHT = RED;

mtype:light WN\_LIGHT = RED;

mtype:light DN\_LIGHT = RED;

mtype:light DE\_LIGHT = RED;

mtype:light NS\_LIGHT = RED;

mtype:light WE\_LIGHT = RED;

// Presence of cars in a given direction: SD, WN, DN, DE, NS, WE

bool SD\_SENSE = false;

bool WN\_SENSE = false;

bool DN\_SENSE = false;

bool DE\_SENSE = false;

bool NS\_SENSE = false;

bool WE\_SENSE = false;

// Locks for direction: SD, WN, DN, DE, NS, WE

bool SD\_LOCK = false;

bool WN\_LOCK = false;

bool DN\_LOCK = false;

bool DE\_LOCK = false;

bool NS\_LOCK = false;

bool WE\_LOCK = false;

// Direction controllers: SD, WN, DN, DE, NS, WE

proctype ControllerSD() {

do

:: (SD\_SENSE && SD\_LIGHT == RED) -> {

atomic { (!WE\_LOCK && !DE\_LOCK && !WN\_LOCK && !DN\_LOCK && !NS\_LOCK); SD\_LOCK = true; };

SD\_LIGHT = GREEN;

};

:: (!SD\_SENSE && SD\_LIGHT == GREEN) -> {

SD\_LIGHT = RED;

SD\_LOCK = false;

};

od;

}

proctype ControllerWN() {

do

:: (WN\_SENSE && WN\_LIGHT == RED) ->

atomic { (!NS\_LOCK && !DE\_LOCK && !SD\_LOCK); WN\_LOCK = true; };

WN\_LIGHT = GREEN;

:: (!WN\_SENSE && WN\_LIGHT == GREEN) -> {

WN\_LIGHT = RED;

WN\_LOCK = false;

};

od;

}

proctype ControllerDN() {

do

:: (DN\_SENSE && DN\_LIGHT == RED) ->

atomic { (!NS\_LOCK && !SD\_LOCK); DN\_LOCK = true; };

DN\_LIGHT = GREEN;

:: (!DN\_SENSE && DN\_LIGHT == GREEN) -> {

DN\_LIGHT = RED;

DN\_LOCK = false;

};

od;

}

proctype ControllerDE() {

do

:: (DE\_SENSE && DE\_LIGHT == RED) ->

atomic { (!NS\_LOCK && !WN\_LOCK && !SD\_LOCK); DE\_LOCK = true; };

DE\_LIGHT = GREEN;

:: (!DE\_SENSE && DE\_LIGHT == GREEN) -> {

DE\_LIGHT = RED;

DE\_LOCK = false;

};

od;

}

proctype ControllerNS() {

do

:: (NS\_SENSE && NS\_LIGHT == RED) ->

atomic { (!SD\_LOCK && !DN\_LOCK && !DE\_LOCK && !WN\_LOCK && !WE\_LOCK); NS\_LOCK = true; };

NS\_LIGHT = GREEN;

:: (!NS\_SENSE && NS\_LIGHT == GREEN) -> {

NS\_LIGHT = RED;

NS\_LOCK = false;

};

od;

}

proctype ControllerWE() {

do

:: (WE\_SENSE && WE\_LIGHT == RED) ->

atomic { (!SD\_LOCK && !NS\_LOCK); WE\_LOCK = true; };

WE\_LIGHT = GREEN;

:: (!WE\_SENSE && WE\_LIGHT == GREEN) -> {

WE\_LIGHT = RED;

WE\_LOCK = false;

};

od;

}

// External environment controller

proctype EnvironmentController() {

do

:: (!SD\_SENSE && SD\_LIGHT == RED) -> SD\_SENSE = true;

:: (!WN\_SENSE && WN\_LIGHT == RED) -> WN\_SENSE = true;

:: (!DN\_SENSE && DN\_LIGHT == RED) -> DN\_SENSE = true;

:: (!DE\_SENSE && DE\_LIGHT == RED) -> DE\_SENSE = true;

:: (!NS\_SENSE && NS\_LIGHT == RED) -> NS\_SENSE = true;

:: (!WE\_SENSE && WE\_LIGHT == RED) -> WE\_SENSE = true;

:: (SD\_SENSE && SD\_LIGHT == GREEN) -> SD\_SENSE = false;

:: (WN\_SENSE && WN\_LIGHT == GREEN) -> WN\_SENSE = false;

:: (DN\_SENSE && DN\_LIGHT == GREEN) -> DN\_SENSE = false;

:: (DE\_SENSE && DE\_LIGHT == GREEN) -> DE\_SENSE = false;

:: (NS\_SENSE && NS\_LIGHT == GREEN) -> NS\_SENSE = false;

:: (WE\_SENSE && WE\_LIGHT == GREEN) -> WE\_SENSE = false;

od;

}

init {

atomic {

run EnvironmentController();

run ControllerSD();

run ControllerWN();

run ControllerDN();

run ControllerDE();

run ControllerNS();

run ControllerWE();

}

}